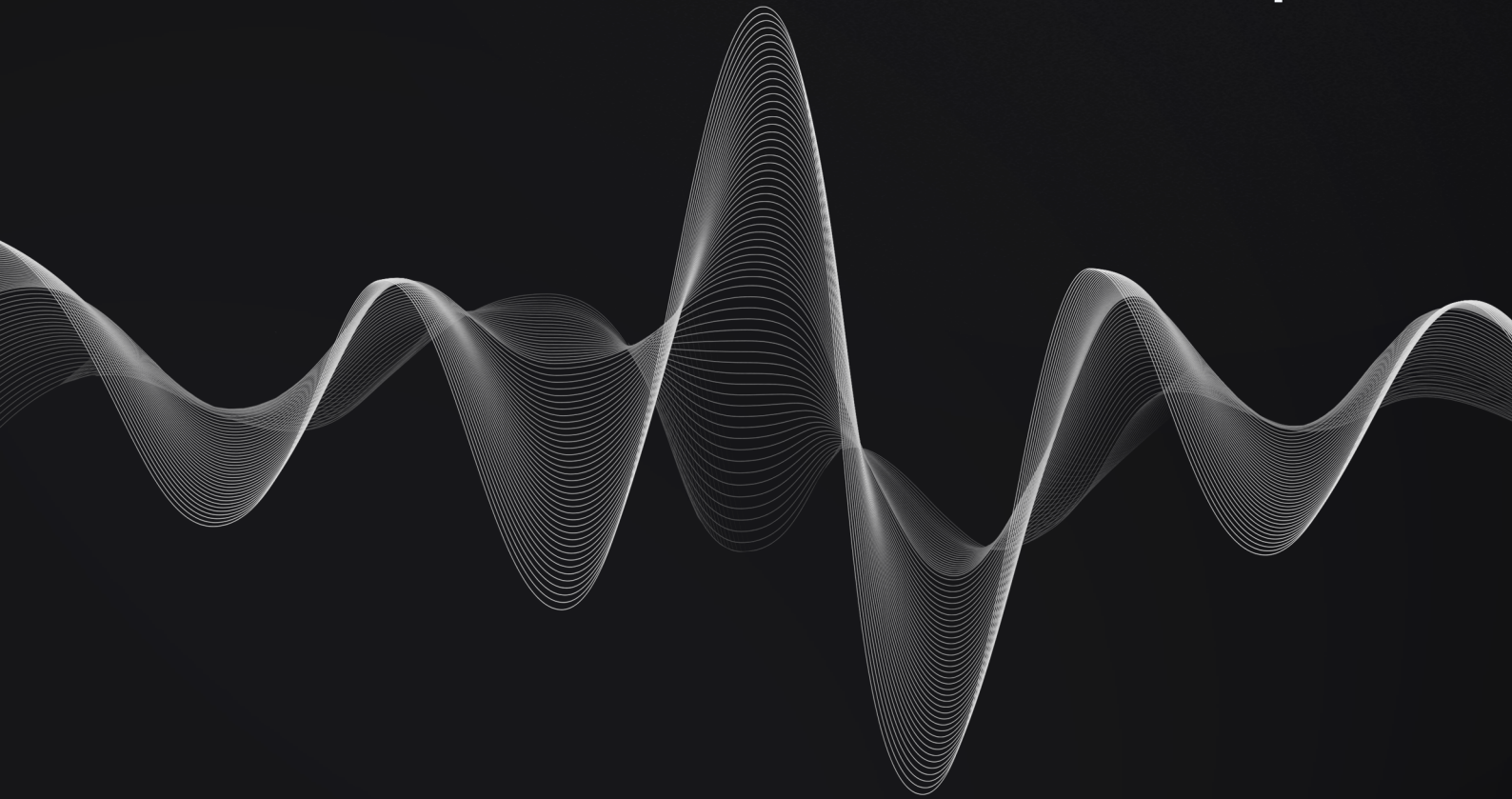




**Hemi**

# Hemi Smart Contract Audit Report




# Document Control

**PUBLIC**

**FINAL**(v2.2)

## Audit\_Report\_HEMI-PRO\_FINAL\_22

Dec 23, 2024		v0.1	Michał Bazyli: Initial draft
Dec 23, 2024		v0.2	Luis Arroyo: Added findings
Dec 24, 2024		v1.0	Charles Dray: Approved
Mar 24, 2025		v2.0	Charles Dray: Finalized
Oct 8, 2025		v2.1	João Simões: Reviewed findings
Oct 9, 2025		v2.2	Charles Dray: Published

<b>Points of Contact</b>	Hemi	Max Sanchez	max@hemi.xyz
	Charles Dray	Resonance	charles@resonance.security
<b>Testing Team</b>	Michał Bazyli	Resonance	michal@resonance.security
	Luis Arroyo	Resonance	luis.arroyo@resonance.security
	João Simões	Resonance	joao@resonance.security

## Copyright and Disclaimer

© 2025 Resonance Security, Inc. All rights reserved.

The information in this report is considered confidential and proprietary by Resonance and is licensed to the recipient solely under the terms of the project statement of work. Reproduction or distribution, in whole or in part, is strictly prohibited without the express written permission of Resonance.

All activities performed by Resonance in connection with this project were carried out in accordance with the project statement of work and agreed-upon project plan. It's important to note that security assessments are time-limited and may depend on information provided by the client, its affiliates, or partners. As such, the findings documented in this report should not be considered a comprehensive list of all security issues, flaws, or defects in the target system or codebase.

Furthermore, it is hereby assumed that all of the risks in electing not to remedy the security issues identified henceforth are sole responsibility of the respective client. The acknowledgement and understanding of the risks which may arise due to failure to remedy the described security issues, waives and releases any claims against Resonance, now known or hereafter known, on account of damage or financial loss.

# Contents

<b>1 Document Control</b>	<b>2</b>
Copyright and Disclaimer .....	2
<b>2 Executive Summary</b>	<b>4</b>
System Overview .....	4
Repository Coverage and Quality.....	4
<b>3 Target</b>	<b>5</b>
<b>4 Methodology</b>	<b>6</b>
Severity Rating.....	7
Repository Coverage and Quality Rating.....	8
<b>5 Findings</b>	<b>9</b>
challengeWithdrawal Reverts For Valid Challenges.....	10
SafeTransfer Should Be Used In Place of Transfer .....	11
Inefficient And Redundant Parameter Handling In createVault Function.....	12
Zero As Parameter.....	13
Floating Pragma .....	14
Variables Initialized to Default Values.....	15
Non-external Function Names (private or internal.) Should Begin With an Underscore.....	16
No Need For == true Or == false Checks .....	17
<b>A Proof of Concepts</b>	<b>18</b>

# Executive Summary

**Hemi** contracted the services of Resonance to conduct a comprehensive security audit of their smart contracts between 13th December, 2024 and 24th Decemer, 2024. The primary objective of the assessment was to identify any potential security vulnerabilities and ensure the correct functioning of smart contract operations.

During the engagement, Resonance allocated 3 engineers to perform the security review. The engineers, including an accomplished professional with extensive proficiency in blockchain and smart-contract security, encompassing specialized skills in advanced penetration testing, and in-depth knowledge of multiple blockchain protocols, devoted 12 days to the project. The project's test targets, overview, and coverage details are available throughout the next sections of the report.

The ultimate goal of the audit was to provide Hemi with a detailed summary of the findings, including any identified vulnerabilities, and recommendations to mitigate any discovered risks. The results of the audit are presented in detail further below.

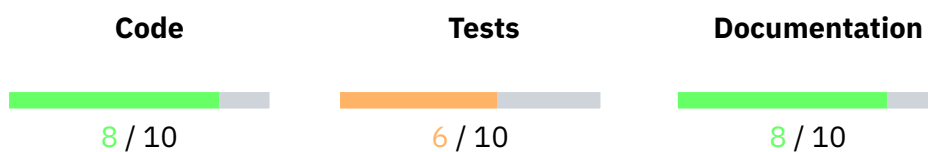


## System Overview

The Hemi Network ("Hemi") is a modular protocol that integrates Bitcoin and Ethereum in a way that amplifies and extends the core capabilities of the two leading blockchain networks.



## Repository Coverage and Quality



Resonance's testing team has assessed the Code, Tests, and Documentation coverage and quality of the system and achieved the following results:

- The code follows development best practices and makes use of known patterns, standard libraries, and language guides. It is easily readable and uses the latest stable version of relevant components. Overall, **code quality is, good**.
- Unit and integration tests are included. The tests cover functional requirements. Overall, **tests coverage and quality is standard**.
- The documentation includes the specification of the system, technical details for the code, relevant explanations of workflows and interactions. Overall, **documentation coverage and quality is good**.

# Target

The objective of this project is to conduct a comprehensive review and security analysis of the smart contracts that are contained within the specified repository.

The following items are included as targets of the security assessment:

- Repository: [hemilabs/bitcoin-tunnel-contracts/contracts](https://github.com/hemilabs/bitcoin-tunnel-contracts/contracts)
- Hash: babb73f5cd72d42fa342103e652e19754a17c323

The following items are excluded:

- External and standard libraries
- Files pertaining to the deployment process
- Financial related attacks

# Methodology

In the context of security audits, Resonance's primary objective is to portray the workflow of a real-world cyber attack against an entity or organization, and document in a report the findings, vulnerabilities, and techniques used by malicious actors. While several approaches can be taken into consideration during the assessment, Resonance's core value comes from the ability to correlate automated and manual analysis of system components and reach a comprehensive understanding and awareness with the customer on security-related issues.

Resonance implements several and extensive verifications based off industry's standards, such as, identification and exploitation of security vulnerabilities both public and proprietary, static and dynamic testing of relevant workflows, adherence and knowledge of security best practices, assurance of system specifications and requirements, and more. Resonance's approach is therefore consistent, credible and essential, for customers to maintain a low degree of risk exposure.

Ultimately, product owners are able to analyze the audit from the perspective of a malicious actor and distinguish where, how, and why security gaps exist in their assets, and mitigate them in a timely fashion.

## Source Code Review - Solidity EVM

During source code reviews for Web3 assets, Resonance includes a specific methodology that better attempts to effectively test the system in check:

1. Review specifications, documentation, and functionalities
2. Assert functionalities work as intended and specified
3. Deploy system in test environment and execute deployment processes and tests
4. Perform automated code review with public and proprietary tools
5. Perform manual code review with several experienced engineers
6. Attempt to discover and exploit security-related findings
7. Examine code quality and adherence to development and security best practices
8. Specify concise recommendations and action items
9. Revise mitigating efforts and validate the security of the system

Additionally and specifically for Solidity EVM audits, the following attack scenarios and tests are recreated by Resonance to guarantee the most thorough coverage of the codebase:

- Reentrancy attacks
- Frontrunning attacks
- Unsafe external calls
- Unsafe third party integrations
- Denial of service
- Access control issues

- Inaccurate business logic implementations
- Incorrect gas usage
- Arithmetic issues
- Unsafe callbacks
- Timestamp dependence
- Mishandled panics, errors and exceptions

## Severity Rating

Security findings identified by Resonance are rated based on a Severity Rating which is, in turn, calculated off the **impact** and **likelihood** of a related security incident taking place. This rating provides a way to capture the principal characteristics of a finding in these two categories and produce a score reflecting its severity. The score can then be translated into a qualitative representation to help customers properly assess and prioritize their vulnerability management processes.

The **impact** of a finding can be categorized in the following levels:

1. Weak - Inconsequential or minimal damage or loss
2. Medium - Temporary or partial damage or loss
3. Strong - Significant or unrecoverable damage or loss

The **likelihood** of a finding can be categorized in the following levels:

1. Unlikely - Requires substantial knowledge or effort or uncontrollable conditions
2. Likely - Requires technical knowledge or no special conditions
3. Very Likely - Requires trivial knowledge or effort or no conditions

		Likelihood		
		Very Likely	Likely	Unlikely
Impact	Strong	Critical	High	Medium
	Medium	High	Medium	Low
	Weak	Medium	Low	Info



# Repository Coverage and Quality Rating

The assessment of Code, Tests, and Documentation coverage and quality is one of many goals of Resonance to maintain a high-level of accountability and excellence in building the Web3 industry. In Resonance it is believed to be paramount that builders start off with a good supporting base, not only development-wise, but also with the different security aspects in mind. A product, well thought out and built right from the start, is inherently a more secure product, and has the potential to be a game-changer for Web3's new generation of blockchains, smart contracts, and dApps.

Accordingly, Resonance implements the evaluation of the code, the tests, and the documentation on a score **from 1 to 10** (1 being the lowest and 10 being the highest) to assess their quality and coverage. In more detail:

- Code should follow development best practices, including usage of known patterns, standard libraries, and language guides. It should be easily readable throughout its structure, completed with relevant comments, and make use of the latest stable version components, which most of the times are naturally more secure.
- Tests should always be included to assess both technical and functional requirements of the system. Unit testing alone does not provide sufficient knowledge about the correct functioning of the code. Integration tests are often where most security issues are found, and should always be included. Furthermore, the tests should cover the entirety of the codebase, making sure no line of code is left unchecked.
- Documentation should provide sufficient knowledge for the users of the system. It is useful for developers and power-users to understand the technical and specification details behind each section of the code, as well as, regular users who need to discern the different functional workflows to interact with the system.



# Findings

During the security audit, several findings were identified to possess a certain degree of security-related weaknesses. These findings, represented by unique IDs, are detailed in this section with relevant information including Severity, Category, Status, Code Section, Description, and Recommendation. Further extensive information may be included in corresponding appendices should it be required.

An overview of all the identified findings is outlined in the table below, where they are sorted by Severity and include a **Remediation Priority** metric asserted by Resonance's Testing Team. This metric characterizes findings as follows:

- ..... **"Quick Win"** Requires little work for a high impact on risk reduction.
- ....|.. **"Standard Fix"** Requires an average amount of work to fully reduce the risk.
- ...||| **"Heavy Project"** Requires extensive work for a low impact on risk reduction.

---

<b>RES-01</b>	challengeWithdrawal Reverts For Valid Challenges	.... ..	Resolved
<b>RES-02</b>	SafeTransfer Should Be Used In Place of Transfer	.... ..	Resolved
<b>RES-03</b>	Inefficient And Redundant Parameter Handling In createVault Function	.....	Acknowledged
<b>RES-04</b>	Zero As Parameter	.... ..	Acknowledged
<b>RES-05</b>	Floating Pragma	.....	Acknowledged
<b>RES-06</b>	Variables Initialized to Default Values	.....	Acknowledged
<b>RES-07</b>	Non-external Function Names (private or internal.) Should Begin With an Underscore	.....	Acknowledged
<b>RES-08</b>	No Need For == true Or == false Checks	.....	Acknowledged



# challengeWithdrawal Reverts For Valid Challenges

High

RES-HEMI-PRO01

Data Validation

Resolved

## Code Section

- `contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol#L1300`
- `contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol#L1300`

## Description

The `SimpleBitcoinVault.challengeWithdrawal()` function challenges a withdrawal by a withdrawer to indicate that a withdrawal was not processed by the vault as expected. If the challenge is successful it will return the number of sats to credit back to the harmed withdrawer along with the withdrawer address these sats should be credited back to.

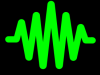
However, the check to determine if a withdrawal has been challenged is not performed correctly, preventing legitimate challenges from not being able to execute this function.

## Recommendation

It is recommended to perform the correct validation in `vaultStateChild.isWithdrawalAlreadyChallenged(uuid)` by checking if it is `false` instead of `true`, as `true` means its already challenged.

## Status

*The issue has been fixed in `dbc7742e307224e1066184ddd232cdd39d96a988`.*



Low

RES-HEMI-PRO02

Data Validation

Resolved

# SafeTransfer Should Be Used In Place of Transfer

## Code Section

- `contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol#L500`

## Description

`SafeTransfer` should be used in place of `Transfer` for Solidity contracts to ensure robust security and error handling. Unlike the basic `Transfer` function, `SafeTransfer` incorporates safeguards against potential smart contract vulnerabilities, such as reentrancy attacks and unexpected token loss.

## Recommendation

it is recommended to use `safeTransfer` instead of `transfer` to send funds from tokens that are not controlled, such as collateral.

## Status

*The issue has been fixed in 6497933c1c4ccedb7f8e44f6f2e5a3552e80ee10.*



# Inefficient And Redundant Parameter Handling In createVault Function

Info

RES-HEMI-PRO03

Data Validation

Acknowledged

## Code Section

- `contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol#L185`

## Description

The `createVault` function is responsible for deploying new vaults via the `IVaultFactory` interface. During testing, it was observed that two parameters, `setupAdmin` and `extraInfo`, are unused in the underlying `SimpleBitcoinVaultFactory` implementation, resulting in redundant input fields that could mislead developers or users. These parameters are not validated, nor do they have any effect on the vault creation process.

## Recommendation

It is recommended to remove unused parameters. If `setupAdmin` and `extraInfo` are not used in the vault creation process, they should be removed from the `createVault` function interface to streamline the logic and reduce confusion.

## Status

*The issue was acknowledged by Hemi's team. The development team stated "This is intentional, as the `setupAdmin` and `extraInfo` are there in the `IVaultFactory` interface for potential future needs of other vault systems, such as when we transition to our `hBitVM` system."*



# Zero As Parameter

Info

RES-HEMI-PR004

Code Quality

Acknowledged

## Code Section

- [contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol#L577](#)
- [contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVaultUTXOLogicHelper.sol#L614](#)
- [contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVaultUTXOLogicHelper.sol#L699](#)

## Description

The use of 0 as a parameter in Solidity function calls can be misleading, especially in the context of financial applications and smart contract logic. While passing zero may not always result in a direct exploit, it can introduce significant ambiguity for developers and users interacting with the contract.

## Recommendation

It is recommended to use explicit named constants (e.g., NO\_FEE, ZERO\_ADDRESS) to make the intent of zero parameters clear in the codebase.

## Status

*The issue was acknowledged by Hemi's team. The development team stated "Low-priority and not addressed."*



# Floating Pragma

Info

RES-HEMI-PR005

Code Quality

Acknowledged

## Code Section

- `contracts/BTCToken.sol#L2`

## Description

Floating pragmas is a feature of Solidity that allows developers to specify a range of compiler versions that can be used to compile the smart contract code. For security purposes specifically, the usage of floating pragmas is discouraged and not recommended. Contracts should be compiled and deployed with a strict pragma, the one which was thoroughly tested the most by developers. Locking the pragma helps ensure that contracts do not accidentally get deployed using too old or too recent compiler versions that might introduce security issues that impact the contract negatively.

It should be noted however that, pragma statements can be allowed to float when a contract is intended for consumption by other developers, as in the case with contracts in a library or EthPM package.

The smart contract of FomoToken makes use of a floating pragma. This is not intended for use as a library for other developers and, as such, should be locked of its pragma.

## Recommendation

It is recommended to lock the pragma of all smart contracts to the same Solidity compiler version.

## Status

*The issue was acknowledged by Hemi's team. The development team stated "Will lock pragma in future versions."*



# Variables Initialized to Default Values

Info

RES-HEMI-PRO06

Gas Optimization

Acknowledged

## Code Section

- [contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol#L790](#)
- [contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol#L948](#)
- [contracts/governance/GlobalConfig.sol#L174](#)

## Description

In Solidity, uninitialized storage variables have a default value. For example, an uninitialized `uint` variable will have a default value of 0.

## Recommendation

It is recommended to avoid initializing variables to its default value.

## Status

*The issue was acknowledged by Hemi's team. The development team stated "Uninitialized storage variables were reviewed and determined to not be an issue but noted for future code cleanliness."*



# Non-external Function Names (private or internal.) Should Begin With an Underscore

Info

RES-HEMI-PRO07

Code Quality

Acknowledged

## Code Section

- `contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol#L630`
- `contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol#L686`
- `contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVault.sol#L854`
- `contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVaultState.sol#L705`
- `contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVaultState.sol#L722`
- `contracts/governance/GlobalConfig.sol#L410`
- `contracts/governance/GlobalConfig.sol#L698`

## Description

According to the Solidity Style Guide, Non-external variable and function names should begin with an underscore.

## Recommendation

It is recommended to rename mentioned functions for clearance and following the Style guides.

## Status

*The issue was acknowledged by Hemi's team. The development team stated "Acknowledged styling suggestions but did not apply change."*





# No Need For `== true` Or `== false` Checks

Info

RES-HEMI-PRO08

Gas Optimization

Acknowledged

## Code Section

- [contracts/BitcoinTunnelManager.sol#L104](#)
- [contracts/vaults/SimpleBitcoinVault/SimpleBitcoinVaultState.sol#L1446](#)

## Description

There is no need to verify that `== true` or `== false` when the variable checked upon is a boolean as well.

## Recommendation

It is recommended to remove `== true` or `== false` from the `requires` if the check parameter is a boolean.

## Status

*The issue was acknowledged by Hemi's team. The development team stated "Noted for future code style improvements."*

# Proof of Concepts

*No Proof-of-Concept was deemed relevant to describe findings in this engagement.*